BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F — A2

(11) International Publication Number: **WO 98/38564**

(43) International Publication Date: 3 September 1998 (03.09.98)

(21) International Application Number: PCT/US98/03752

(22) International Filing Date: 24 February 1998 (24.02.98)

(30) Priority Data:
60/039,230 28 February 1997 (28.02.97) US

(71) Applicant *(for all designated States except US)*: SIEBEL SYSTEMS, INC. [US/US]; 1855 South Grant Street, San Mateo, CA 94402 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: BRODERSEN, Robert, S. [US/US]; 17 Spinaker Drive, Redwood City, CA 94065 (US). CHATTERJEE, Prashant [US/US]; 21281 Glenmont Drive, Saratoga, CA 95070 (US). LIM, Peter, S. [US/US]; 917 Governors Bay Drive, Redwood City, CA 94065 (US).

(74) Agents: GOLDMAN, Richard, M.; Cooley Godward LLP, Five Palo Alto Square, 3000 El Camino Real, Palo Alto, CA 94306–2155 (US) et al.

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, GW, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: PARTIALLY REPLICATED DISTRIBUTED DATABASE WITH MULTIPLE LEVELS OF REMOTE CLIENTS

(57) Abstract

A method of and system for collecting, storing, and retrieving data in a database management system. The database management system includes a master database server (4), at least one workgroup server (315), and a plurality of workgroup user clients (310). The workgroup server (315) is interposed between the master database server (4) and said workgroup user clients (310). The method creating a transaction in a local database resident on one of the workgroup user clients (310), entering the transaction into a transaction log resident on the workgroup user client (310), and creating a transaction file corresponding to the transaction in an outbox of said workgroup user client (310). Next, the transaction file is copied to an inbox identified to the workgroup user client (310) and updating the transaction file into a workgroup database (305) resident on the workgroup server (315). The workgroup database (305) includes a transaction log. Finally, the workgroup database (305) is read into the transaction log, skipping those transactions which originate at the master database server (4), and creating data files corresponding to the agency workgroup database. Data files corresponding to transactions originating at the workgroup user client (310) are copied to an inbox on the master database server (4). This inbox corresponds to the workgroup server (315). The transactions are copied into a master database (3) on the master database server (4).

PARTIALLY REPLICATED DISTRIBUTED DATABASE WITH MULTIPLE LEVELS OF REMOTE CLIENTS

INTRODUCTION

I. Technical Field

This invention relates to a system and method for providing updates to a network of partially replicated relational database systems, and, more particularly, for providing an efficient means for computing the visibility to a client on the network of a transaction processed against the database.

II. Background

Relational databases are a commonly-employed data structure for representing data in a business or other environment. A relational database represents data in the form of a collection of two-dimensional tables. Each table comprises a series of cells arranged in rows and columns. Typically, a row in a table represents a particular observation. A column represents either a data field or a pointer to a row in another table.

For example, a database describing an organizational structure may have one table to describe each position in the organization, and another table to describe each employee in the organization. The employee table may include information specific to the employee, such as name, employee number, age, salary, etc. The position table may include information specific to the position, such as the position title ("salesman", "vice president", etc.), a salary range, and the like. The tables may be related by, for example, providing in each row of the employee table a pointer to a particular row in the position table, coordinated so that, for each row in the employee table, there is a pointer to the particular row in the position table that describes that

replicated database is not made to the central database, leading to a discrepancy between the information that is stored in the replicated copy of the database and the information that is stored in the central database. Although it is possible to journal modifications made to the replicated copy and apply an identical modification to the central database, one problem that this approach faces is the possibility of colliding updates; that is, where a user of a replicated copy makes a change to data that is also changed by a user of the central copy or by the user of another replicated copy.

It is therefore desirable to provide a capability to maintain one or more partially-replicated copies of a central database, in such a way that the degree of replication may be easily changed without requiring a refresh of the entire replicated database, and that permits updates to be coordinated among users of the central database and users of the partially replicated databases.

SUMMARY OF THE INVENTION

The present invention is directed to a method of maintaining a partially replicated database in such a way that updates made to a central database, or to another partially replicated database, are selectively propagated to the partially replicated database. Updates are propagated to a partially replicated database if the owner of the partially replicated database is deemed to have visibility to the data being updated. Visibility is determined by use of predetermined rules stored in a rules database. In one aspect of the invention, the stored rules are assessed against data content of various tables that make up a logical entity, known as a docking object, that is being updated.

In another aspect of the invention, the stored rules are assessed against data content of one or more docking objects that are not necessarily updated, but that are related to a docking object being updated. In one embodiment, the visibility attributes of the related docking objects are recursively determined.

In yet another aspect of the invention, changes in visibility are determined to enable the central computer to direct the nodes to insert the docking object into its

workgroup connected clients (330-a). In this embodiment of the invention the workgroup server (315) is directly connected to the workgroup connected clients (330-a). The method of this embodiment of our invention includes creating a transaction in a local database resident on one of the workgroup connected clients (330-a), entering the transaction into a transaction log resident on the workgroup connected client (330-a), and creating a transaction file corresponding to the transaction in an outbox of the workgroup connected client (330-a). In this embodiment of our invention, the next step is copying the transaction file to an inbox identified to the workgroup connected client (330-a) and updating the transaction file into a workgroup database (305) resident on the workgroup server (315), where the workgroup database (305) includes a transaction log. The transactions are directly entered into the transaction log in the workgroup server (315).

A still further embodiment of our invention is its incorporation into an article of manufacture, that is, a disk, a tape, or the like. The article is a computer usable, i.e., readable, medium having computer readable program code for collecting, storing, and retrieving data in a database management system. The database management is one, as described above, having a master database server (4), an application server (303), at least one workgroup server (315), and a plurality of workgroup user clients (310), where the workgroup server (315) is interposed between the master database server (4) and the workgroup user clients (310). The computer readable program in the article of manufacture includes computer readable program code for causing a computer to create a transaction in a local database resident on one or more of the individual workgroup user clients (310),and entering the transaction into a transaction log resident on one of the workgroup user clients (310), that is, the workgroup user client (310) where the transaction originated, and creating a transaction file corresponding to the transaction in an outbox of the workgroup user client (310). In this embodiment of our invention the computer readable program code causing the computer to effect copying the transaction file to an inbox identified to the workgroup user client (310) and updating the transaction file into a workgroup database (305) resident on the workgroup server (315). The workgroup database (305) includes a transaction log. Finally, the computer readable program code causes the computer to effect reading the workgroup database (305)

present invention.

Figure 2 depicts a database schema that shows the relationship of the various components that make up a Docking Object.

Figure 3 depicts steps performed by an update manager to update a database.

Figure 4 depicts steps performed by a Docking Manager to transmit and/or receive one or more transaction logs.

Figure 5 depicts the steps performed by a merge processor to merge transaction log records into an existing database.

Figure 6 depicts the steps performed by a log manager to prepare a partial transaction log.

Figure 7 depicts the steps performed by a visibility calculator for calculating visibility for a docking object as invoked by a log manager.

Figure 8 depicts the steps performed to synchronize a partially replicated database in response to a change in data visibility.

Figure 9 depicts the logical database configured to support multi-user docking clients.

Figure 10 depicts a database diagram describing the database design to support multi-user docking clients.

DESCRIPTION OF SPECIFIC EMBODIMENTS

Overview

Figure 1 depicts an overview of the operation of one embodiment of the present invention. Figure 1 depicts a central computer system 1 and three remote

At some point in time, at the convenience of the operator of central computer system 1, log manager 9 is activated. Log manager 9 takes as input central update log 15 and produces as output a set of partial logs 17-a, 17-b and 17-c according to visibility rules as will be further described herein. Each of partial logs 17-a, 17-b and 17-c corresponds to one of nodes 21-a, 21-b and 21-c. When a node docking manager such as node docking manager 25-a enters into communication with central docking manager 5 and optionally requests transmission of its corresponding partial log, central docking manager 5 takes as input the appropriate partial log, such as partial log 17-a, and presents it to node docking manager 25-a. Node docking manager 25-a then replicates partial log 17-a as merge log 37-a.

At some point in the future, at the convenience of the operator of node 21-a, merge processor 27-a is activated. Merge processor 27-a takes as input merge log 37-a, and applies the updates described therein to partially replicated database 23-a.

In addition to node 21-a, Figure 1 also depicts two additional nodes 21-b and 21-c. Node 21-b is depicted in communication with central computer 1. However, unlike node 21-a, the operator of node 21-b has requested only to send his updates to central computer system 1, and has not requested to be presented with changes made elsewhere to be made to his partially replicated database 23-b. This may be, for example, if the operator has an urgent update that must be made as soon as possible, but does not have the time to receive updates from other nodes. Accordingly, Figure 1 shows only transmission of node update log 35-a from node docking manager 25-b to central docking manager 5, and no transmission from central docking manager 5 to node docking manager 25-b. Accordingly, the merge manager for node 21-b is not activated and is not shown.

Likewise, node 21-c is depicted as not in communication with central computer system 1. Accordingly, the docking manager for node 21-c is not activated and is not shown.

By the cycle described above, updates made by each of nodes 21-a, 21-b and

of a Docking Object is "visible" to a particular node 21. If a Docking Object is visible to a particular node, that node will receive updates for data in the Docking Object. Visibility Rules are of two types, depending on the field RULE_TYPE. A Visibility Rule with a RULE_TYPE of "R" is referred to as an SQL Rule. An SQL Rule includes a set of Structured Query Language (SQL) statements that is evaluated to determine if any data meeting the criteria specified in the SQL statements exists in the Docking Object. If so, the Docking Object is visible to the node. A Visibility Rule with a RULE_TYPE of "O" is referred to as a Docking Object Rule. A Docking Object Rule specifies another Docking Object to be queried for visibility. If the specified Docking Object is visible, then the Docking Object pointing to it is also visible.

A Related Docking Object is a Docking Object that is propagated or deleted when the Docking Object under consideration is propagated or deleted. For example, an Opportunity Docking Object may have related Docking Objects representing the sales contacts, the organizations, the products to be sold, and the activities needed to pursue the opportunity. When an Opportunity Docking Object is propagated from Central Database 3 to one of node databases 23, the related docking objects are also propagated.

Figure 2 depicts a database schema that shows the relationship of the various components that make up a Docking Object. The schema is a meta-database, in that it does not describe the data being accessed in the database. Rather, the schema is a separate database that defines the structure of the database being accessed. That is, it is a database comprising tables that describe the relationships and data contexts of another database.

Each of the tables shown in Figure 2 is a table in a relational database, and as such is in row-column form. Many columns represent fields that are common to all the illustrated tables. Such fields include for example, a ROW_ID to identify a particular row in the table, as well as fields to tack the date and time that a row was created and last modified, and the identity of the user who created or modified the row. In addition, each table contains fields specific to that table, and which are

Table S_DOBJ_VIS_RULE 71 contains the visibility rules associated with a particular Docking Object. S_DOBJ_VIS_RULE 71 contains the fields DOBJ_ID, RULE_SEQUENCE, RULE_TYPE, SQL_STATEMENT and CHECK_DOBJ_ID. Field DOBJ_ID identifies the Docking Object with which a particular visibility rule is associated. Field RULE_SEQUENCE is a sequence number that indicates the sequence, relative to other visibility rules in table S_DOBJ_VIS_RULE 71, in which the particular visibility rule should be run. RULE_TYPE specifies whether the particular visibility rule is of type "R," indicating an SQL visibility rule or of type "O," indicating a Docking Object visibility rule.

If RULE_TYPE is equal to "R," field CHECK_DOBJ_ID is not meaningful, and field SQL_STATEMENT contains an SQL statement that is evaluated using the Primary ROW-ID of the primary table associated with this Docking Object and a particular Node 21. If the SQL statement returns any records, the Docking Object is deemed to be visible to the Node 21 for which visibility is being determined.

If RULE_TYPE is equal to "O," both field CHECK_DOBJ_ID and field SQL_STATEMENT are meaningful. Field CHECK_DOBJ_ID specifies a docking object whose visibility should be determined. If the specified docking object is deemed to be visible, then the docking object associated with the visibility rule is also visible. Field SQL_STATEMENT contains a SQL statement that, when executed, returns the Row-ID of the docking object identified by CHECK_DOBJ_ID that corresponds to the docking object instance associated with the visibility rule.

Table S_APP_TBL 73 is an Application Table that describes all the tables used in a particular application. It is pointed to by table S_DOBJ_TBL 69 for each member table in a docking object, and by table S_DOBJ for the primary table in a docking object. S_APP_TBL 73 points to table S_APP_COL 75, which is an Application Column Table that describes the columns of data in a particular application. S_APP_TBL 73 points to table S_APP_COL 75 directly through a primary key and indirectly through such means as a Foreign Key Column Table 81, User Key Column Table 83, and Column Group Table 85. The relationship of an Application Table, Application Column Table, Foreign Key Column Table, User Key

An update manager 11 executing in central computer system 1 operates in an analogous manner, except that it updates central database 3 and writes its log records to central update log 11.

Docking Processing

Figure 4 depicts steps performed by a Docking Manager 25 such as Docking Manager 25-a, 25-b or 25-c to transmit and/or receive one or more transaction logs. Docking Manager 25 is invoked by the user of a remote node such as node 21-a, 21-b or 21-c, whereby the user requests that the node dock with central computer 1 to upload an update log such as update log 35-a to central computer 1, to download a partial log such as partial log 17-a, or both. Execution of Docking Manager 25 begins in step 121. In step 123, Docking Manager 25 connects with central computer 1 under the control of Central Docking Manager 5. This connection can be any connection that enables data exchange. It is anticipated that the most common form of a connection is a telephone line used in conjunction with a modem, but other forms of data connection, such as a Local Area Network or a TCP/IP connection may also be used. Step 125 checks to see whether the user has requested that node update log 35-a be uploaded to the Central Computer 1. If so, execution proceeds to step 127. If not, step 127 is skipped and control is given to step 129. In step 127, Docking Manager 25 uploads its update log to central computer 1. The upload may be accomplished with any known file transfer means, such as XMODEM, ZMODEM, KERMIT, FTP, ASCII transfer, or any other method of transmitting data. In step 129, Docking Manager 25 checks to see whether the user has requested that a partial log such as partial log 17-a be downloaded from Central Computer 1. If so, execution proceeds to step 131. If not, step 131 is skipped and control is given to step 133. In step 131, Docking Manager 25 downloads its partial log from central computer 1. The download may be accomplished with any known file transfer means, such as XMODEM, ZMODEM, KERMIT, FTP, ASCII transfer, or any other method of transmitting data. In step 133, having completed the requested data transfer, Docking Manager 25 exits.

Merge Processing

Merge processing is performed by a processor such as node merge processor

In the above example, when the transaction for node 27-b was presented to merge processor 7, merge processor 7 would compare "Keith Palmer," the prior value of the field as recorded by node 27-b to "Carl Lake," the present value of the field as recorded in central database 3. Detecting the mismatch, merge processor 7 may then generate a transaction to change the value "Greg Emerson" to "Carl Lake," and write that transaction to update log 15. In a subsequent docking operation, that transaction would be routed back to node 27-b to bring its database 23-b in synch with the other databases.

The above is one example of a collision and a resulting corrective action. Other types of collisions include, for example, an update to a row that has previously been deleted, inserting a row that has previously been inserted, and the like. Merge processing must detect and correct each of these collisions. This may be performed using any of a number of well-known methods, and is not discussed further.

Figure 5 depicts the steps performed by merge processor such as central merge processor 7. Although it depicts merge processor 7 writing to central database 3 and to transaction log 15, it is equally representative of a node merge processor such as node merge processor 27-a, 27-b or 27-c updating a node database 23-a, 23-b or 23-c. Merge processing begins at step 141. In step 143, merge processor 7 finds the first unprocessed transaction on received log 19. In step 147, merge processor 7 selects a transaction from received log 19. In step 149, merge processor 149 attempts to update database 3 according to the transaction selected in step 147. In step 151, merge processor 7 determines whether the database update of step 149 failed due to a collision. If so, merge processor proceeds to step 153, which generates a corrective transaction. Following the generation of the corrective transaction, the merge processor returns to step 149 and again attempts to update database 3. If no collision was detected in step 151, execution proceeds to step 157. In step 157, merge processing checks to see if it is executing on central computer 1. If so, step 155 is executed to journal the transaction to log 15. In any case, either if step 157 determines that the merge processing is being performed on a node or after step 155, execution proceeds to step 159. Step 159 checks to see if any transactions remain to be processed from log 19. If so, execution repeats from step 147, where the next transaction is selected. If not, merge processing exits in step 161.

data stored in the schema depicted in Figure 2, to determine whether a particular transaction that updates a particular row of a particular docking object is visible to a particular node.

The Visibility calculator begins execution at step 201. In step 203, the visibility calculator makes a default finding that the transaction is not visible. Therefore, unless the visibility calculator determines that a transaction is visible, it will exit with a finding of no visibility. In step 205, the visibility calculator selects the first visibility rule associated with the docking object. This is done by finding the table S_DOBJ_VIS_RULE 71 associated with the current Docking Object as pointed to by table S_DOBJ 61. In step 205, the visibility calculator selects the row of table S_DOBJ_VIS_RULE 71 with the lowest value for field RULE_SEQUENCE.

In step 207, the Visibility Calculator checks the field RULE_TYPE for a value of "R." The value of "R" indicates that the rule is a SQL visibility rule. If so, the Visibility Calculator proceeds to step 209. In step 209 the Visibility Calculator obtains a SQL statement from field SQL_STATEMENT and executes it. An example of such an SQL statement might be:

```
SELECT 'X' FROM S_OPTY_EMP
  WHERE OPTY_ID = :PrimaryRowId
    AND EMP_ID = :NodeId;
```

This SQL statement causes a query to be made of application table S_OPTY_EMP. The query selects any records meeting two criteria. First, the records selected must have a field OPTY_ID, which is a row id or key, equal to the Primary Row-ID of the Docking Object whose visibility is being determined. Second, the records selected must have a field EMP_ID, which may be for example, an identifier of a particular employee, equal to the NodeId of the node for whom visibility is being determined. In ordinary language, this SQL statement will return records only if a row is found in a table that matches employees to opportunities, where the opportunity is equal to the one being updated, and the employee to whom the opportunity is assigned is the operator of the node.

This is a simplistic example, provided for maximum comprehension. More complex SQL statements are possible. For example, the rule:

```
SELECT 'X' FROM
&Table_Owner.S_ACCT_POSTN ap
```

step 243, the Visibility Calculator references the visibility just calculated for a docking object. If the Docking Object is visible, execution proceeds to step 245. Step 245 references the S_DOBJ_INST table, to verify that a row exists for the Docking Object for the current node. If a row exists, this indicates that the node in question already has a copy of the referenced Docking Object, and the routine proceeds to step 255, where it exits. If, however, no row exists for the Docking Object at the node being processes, this indicates that the node in question does not have a copy of the Docking Object on its partially replicated database. The routine then proceeds to step 247, where a transaction is generated to direct the node to insert the Docking Object into its partially replicated database.

If step 243 determines that the Docking Object is not visible, execution proceeds to step 249. Step 249 references the S_DOBJ_INST table, to verify that no row exists for the Docking Object for the current node. If step 243 determines that no row exists in the S_DOBJ_INST table for the current docking object for the current row, this indicates that the node in question does not have a copy of the referenced Docking Object, and the routine proceeds to step 255, where it exits. If, however, a row exists for the Docking Object at the node being processed, this indicates that the node in question does have a copy of the Docking Object on its partially replicated database. The routine then proceeds to step 251, where a transaction is generated to direct the node to delete the Docking Object from its partially replicated database.

Referring again to Figure 7, following the data synchronization routine of step 228, the Visibility Calculator proceeds to step 229, where it exits. Referring to Figure 6, as previously described, the resulting finding of visibility is available to be checked by the log manager in step 183 to determine to write the transaction.

Referring again to figure 7, if step 211 determines that no records were returned by the execution of the SQL statement in step 209, execution proceeds with step 215. Step 215 checks to see whether there are any remaining visibility rules to be assessed. If not, the visibility calculator proceeds to step 228 to synchronize the database, and then to step 229, where it exits. In this case, the default mark of no visibility that was set in step 203 remains set. This value will also be used by the log manager as shown in Figure 6, step 183, to determine not to write the transaction.

Calculator recursively invokes itself to determine visibility of the related docking object. The recursively invoked Visibility Calculator operates in the same manner as the Visibility Calculator as called from the Log Manager 9, including the capability to further recursively invoke itself. When the recursive call concludes, it returns a visibility indicator for the related Docking Object, and control proceeds to step 227. In step 227, the Visibility calculator determines whether the related Docking Object was determined to have been visible. If so, the Visibility Calculator proceeds to step 213 to mark the originally current Docking Object as visible, and then to step 228 to synchronize the database and then to step 229 to exit. If the related Docking Object was not determined to be visible, control proceeds to step 215 to determine whether additional visibility rules remain to be assessed.

The Visibility Calculator, in conjunction with the Log Manager is therefore able to determine what subset of update transaction data is required to be routed to any particular node. This operation serves to reduce the transmission of unneeded data from the Central Computer 1 to the various nodes such as nodes 21-a, 21-b and 21-c that utilize partially replicated databases, and to reduce the system resources such as disk space needed to store, and the CPU time needed to process, what would otherwise be required to maintain a fully replicated database on each remote node.

The operation of the log manager 9 in conjunction with the Visibility Calculator herein described will be apparent from reference to the description and to the drawings. However, as a further aid in the description of these facilities, a pseudocode representation of these facilities is hereto attached as an Appendix.

Multiple-User Docking Clients

The present invention may be enhanced by adding support for multi-user docking clients. This capability extends the docking architecture to permit replication of database information from the master database server to a variety of geographically dispersed workgroup servers (also called agency database servers). This capability allows multiple users to connect to these agency database servers. Mobile users need to synchronize their local databases against the master database server.

Multi-user docking clients provide the basis of a single, integrated, logical database. Figure 9 depicts the logical database configured to support multi-user docking clients. A single master database 3 at headquarters routes transactions to

(315), where the workgroup database (305) includes a transaction log. The transactions are directly entered into the transaction log in the workgroup server (315).

Multi-user docking clients comprise an agency server (315) (or simply, "agency"), running a workgroup database 305, and one or more workgroup users (310) connected to the server via a LAN or other connection. Agency server (315) may be a Windows/NT server or other server. Multi-user docking clients behave in the same way as single-user mobile clients. In addition, multi-user docking clients store data for one or many users; allow multiple users to access and change data on the workgroup database simultaneously; permit users to execute server-side programs against the workgroup; and execute a periodic docking program to exchange data with the master database at predefined times or intervals.

In this aspect of the invention, the database is configured to support a plurality of users in a single docking entity. More particularly, one aspect of the invention is a method of collecting, storing, and retrieving data in a data base management system having a master database server (4), at least one workgroup server (315), and a plurality of workgroup user clients (310). In this embodiment of the invention the workgroup server (315) is interposed between the master database server (4) and said workgroup user clients (310). The method of this embodiment of our invention includes creating a transaction in a local database resident on one of the workgroup user clients (310), entering the transaction into a transaction log resident on the workgroup user client (310), and creating a transaction file corresponding to the transaction in an outbox of the workgroup user client (310). In this embodiment of our invention, the next step is copying the transaction file to an inbox identified to the workgroup user client (310) and updating the transaction file into a workgroup database (305) resident on the workgroup server (315), where the workgroup database (305) includes a transaction log. The next step in the method of our invention includes reading the workgroup database (305) transaction log , skipping those transactions which originate at the master database server (4) so as to avoid looping, and creating data files corresponding to the entries in the transaction log. These entries are copied to an inbox on the master database server (4) which corresponds to entries on the workgroup server (315). These entries are used to update the transactions into a

(303), at least one workgroup server (315), and a plurality of workgroup user clients (310), where the application server (303) and the workgroup server (315) are interposed between the master database server (4) and said workgroup user clients (310). In this embodiment the code causes the database management system resident on a workgroup client to create a transaction in a local database resident on the workgroup user clients (310), enter the transaction into a transaction log resident on the workgroup user client (310), and create a transaction file corresponding thereto in an outbox of said workgroup user client (310). Next, the transaction file is caused to be copied to an inbox identified to the workgroup user client (310) and the transaction file is updated into a workgroup database (305) resident on the workgroup server(315). To be noted is that the workgroup database (305) includes a transaction log. Next, the software reads the workgroup database (305) transaction log, skipping those transactions which originated at the master database server (4), that is, to avoid looping, creating data files corresponding to the entries in the transaction log, and copying data files corresponding to transactions originating at the workgroup user client (310) to an inbox on the master database server (4) corresponding to the workgroup server (315), and updating the transactions into a master database (3) on the master database server (4).

The following flow descriptions describe the process and flow of transactions and correspondence among the various components.

In order to process a transaction from the agency to the central database at the headquarters node, the periodic docker reads the transaction log in the agency database and creates a dx file in the agency node's outbox. The periodic docker checks the originating node of the transaction entry read from the transaction log and skips those which originate at the headquarters node, or the parent node of the agency node, in configurations having multiple agency levels. This is required because the periodic docker turns on transaction logging when calling datamerge to merge the changes from the HQ node. Transaction logging is needed for the log manager to route those transactions that must be routed to client nodes down below, but not routed back to the headquarters node. Otherwise, an infinite loop could occur. The periodic docker updates the docking status table. This is required so that the log

with the workgroup server (315). A single mobile user (21-a) can access a single-user database (23-a).

In order to process a transaction from an agency to a client, the log manager executing on the agency reads the transaction from the transaction log, checks whether that transaction is visible to the mobile client. If so, the log manager writes the corresponding dx files into the client node=s outbox on the agency node. Subsequently, the client node docks with agency node. The docking manager reads the dx files from its outbox and copies them down to the inbox on the client node. The docking manager then calls datamerge to merge these transaction records into the client database. No transaction is logged.

Figure 10 depicts a database diagram describing the database design to support multi-user docking clients. Node table 65 (named S_NODE) has a one-to-many relationship to node relationship table 360 (named S_NODE_REL), node employees table 365 (named S_NODE_EMP), dock object instance table 370 (named S_DOCK_INST, equivalent to S_DOBJ_INST table 63) and dock status table 375. Node employees table 365 has a many-to-one relationship with employees table 380 (named S_EMPLOYEE)

Node employees table 365 serves as an intersection table between node table 65 and employees table 380. It includes the following fields.

Column	Type	Constraints
ROW_ID	VARCHAR2(15)	Primary not null
CREATED	DATETIME	not null
CREATED_BY	VARCHAR2(15)	not null
LAST_UPD	DATETIME	not null
LAST_UPD_BY	VARCHAR2(15)	not null
MODIFICATION_NUM	NUMBER	not null
CONFLICT_ID	VARCHAR2(15)	not null
NODE_ID	VARCHAR2(15)	$Unique_1$ ref(Node) not null
EMP_ID	VARCHAR2(15)	$Unique_2$ ref(Emp) not null

Node employees table 365 is indexed as follows:

unique index S_NODE_EMP_P1 on S_NODE_EMP (ROW_ID)

LAST_UPD_BY	VARCHAR2(15)	not null
MODIFICATION_NUM	NUMBER	not null
CONFLICT_ID	VARCHAR2(15)	not null
NODE_ID	VARCHAR2(15)	not null
SUB_NODE_ID	VARCHAR2(15)	not null
RELATION_TYPE	VARCHAR2(30)	not null

Node relationship table 360 is indexed as follows:

unique index S_NODE_REL_P1 on S_NODE_REL (ROW_ID)

unique index S_NODE_REL_U1 on S_NODE_REL
(NODE_ID,SUB_NODE_ID,RELATION_TYPE,CONFLICT_ID)

non-unique index S_NODE_REL_F2 on S_NODE_REL (SUB_NODE_ID)

CONCLUSION

Various modifications to these embodiments will be readily apparent to those skilled in the art, and the generic principles defined herein may be applied to other embodiments without the use of inventive faculty. Thus, the present invention is not intended to be limited to the embodiments shown herein, but is to be accorded the widest scope consistent with the principles and novel features disclosed herein.

All publications and patent applications mentioned in this specification are herein incorporated by reference to the same extent as if each individual publication or patent application was specifically and individually indicated to be incorporated by reference.

The invention now being fully described, it will be apparent to one of ordinary skill in the art that many changes and modifications can be made thereto without departing therefrom.

```
      Goto next transaction
    END IF;

    -- Process all other types of transactions

    -- This is the visibility calculator!
    -- This routine also processes implicit visibility events
    -- Later: Data Merge can call this function to check
whether a txn is
    --    still visible when merging txns into a laptop or
server database.
    CheckVisibility (LaptopNodeId, LogRecordType, TableName,
TransRowId);
    IF txn is visible THEN
      -- Write transactions to UserTxnLog file depending on
the
      -- type of LogRecordType.
      Write the txn to the user log file
      ++NumBatchTxns
    END IF;

    -- Finished processing the txn
    -- Commit (if needed)
    IF NumBatchTxns = MaxBatchTxns THEN
      -- Assume that separate process comes around and
deletes
      -- Txns in S_DOCK_TRANSACTION_LOG that have been
processed
      -- for all nodes. So, no need to delete the txns from
the log.
      Update last LOG_EXTRACT number for Laptop in
S_DOCK_STATUS
      Commit;
      NumBatchTxns = 0
    END IF;

    ++NumTxns
End Loop; /* Each transaction in the Txn Log table */

-- Commit
Update last LOG_EXTRACT number for Laptop in S_DOCK_STATUS
Commit;

-- Close log file (if needed)
IF UserTxnLogFileP != NULL THEN
  Close File;
END IF;

StopDictApi ();
```

Check Visibility Routines

```
-- Check if a record in the txn log is visible to a
LaptopNodeId
BOOL   CheckVisibility   (LaptopNodeId,   LogRecordType,
```

```
PrimaryRowId)
      IF object is visible THEN
       -- Because CheckObjectVisibility() also processes
implicit
       -- visibility events, we must loop through ALL
docking objects
        -- even if we already know that the Txn is visible.
       -- Exception: if the table has VIS_event_FLG = 'N'
       --   then we can return immediately.
       IF Table.visibilityEventFLG = 'N' THEN
         return TRUE;
       ELSE
         bVisible = TRUE;
       END IF;
     END IF;
   END LOOP;
 END LOOP;

 return bVisible;
}

-- Check if an instance of a docking object is visible to
the laptop user.
-- Also processes implicit visibility events!
BOOL CheckObjectVisibility    (LaptopNodeId,
DockingObjectName, PrimaryRowId)
{
  FOR each visibility rule for the Docking Object LOOP
    IF RuleType = RuleSQL THEN
      Run the select SQL statement using PrimaryRowId;
      IF any rows returned THEN
        row is visible
        -- Process an implicit Download Object
        DownloadObjectInstance    (LaptopNodeId,
PrimaryTableName,
                                   PrimaryRowId);
        return TRUE;
      END IF;
    ELSIF RuleType = CheckDockingObject THEN
      Run the ParameterSQL using PrimaryRowId to get
newPrimaryRowId
      FOR each record retrieved by ParameterSQL LOOP
        -- RECURSIVE!
        CheckObjectVisibility   (LaptopNodeId,
CheckDockingObjectName,
                                 newPrimaryRowId);
        IF rc = TRUE THEN
          -- Process an implicit Download Object
          DownloadObjectInstance    (LaptopNodeId,
PrimaryTableName,
                                     PrimaryRowId);
          return TRUE;
        END IF;
      END LOOP;
```

```
struct
{
  CHAR* selectList;
  CHAR* fromClause;
  CHAR* whereClause;
  UINT numTables; /* also the number of joint to reach the
Primary Table */
} GenStmt;


GeneratePrimaryIdSQL (Table, DockingObject)
{
  /* there may be more than one SQL statement, so we have
a dynamic
     array of SQL statements. Each element in the array is
a path
     from the Table to the Primary Table*/
  DynArrId GenStmtArr;
  GenStmt newGenStmt;

  CHAR* sqlStmt;

  DynArrCreate (GenStmtArr);

  -- Create the first element and initialize
  newGenStmt = malloc();
  newGenStmt.numTables = 1;
  newGenStmt.selectList = "SELECT row_id";
  newGenStmt.fromClause = "FROM <Table> t1";
  newGenStmt.whereClause = "WHERE t1.ROW_ID = :row_id";
  DynArrAppend (GenStmtArr, &newGenStmt);

  /* Recursively follow FKs to the PrimaryTable */
     Build the select, from and where clause simultaneously
*/
  AddPKTable (Table, DockingObject, GenStmtArr, 0);

  -- Union all the paths together
  numStmts = DynArrSize (GenStmtArr);
  FOR all elements in the array LOOP
     tmpSqlStmt    =    GenStmtArr[j].selectList||
GenStmtArr[j].fromClause ||
 GenStmtArr[j].whereClause;
    sqlStmt = sqlStmt || 'UNION' || tmpSqlStmt;
  END LOOP;

  DynArrDestroy (GenStmtArr);

  IF sqlStmt = NULL THEN
    Error: no path from Table to Primary Table.
  END IF;
}


-- Recursively follow all FKs to the Primary Table
```

```
      -- Only count FKs to other member tables in the same
Docking Object
      ++numFKs;

    END IF;
  END LOOP;

  RETURN;
}
```

Process Visibility Events

```
-- Download an Object Instance to a Laptop
-- This function also downloads all Related Docking Object
instances.
BOOL DownloadObjectInstance (LaptopNodeId, ObjectName,
PrimaryRowId)
{
  -- Check if the object instance is already downloaded to
the laptop
  Find the object instance in the S_DOBJ_INST table
  IF exists on laptop THEN
    return TRUE;
  END IF;

  -- Register object instance in S_DOBJ_INST table

  -- Write Download Object records to the Txn Log
  FOR each member table of the docking object LOOP
    Generate SQL select statement to download records
    Write each retrieved record to the User Txn Log file
  END LOOP;

  -- Download records for Parent Object instances
  FOR each RelatedDockingObject LOOP
    Run ParameterSQL to get newPrimaryId of
RelatedDockingObjects
    FOR each newPrimaryId retrieved LOOP
      -- Check if the instance of the object is visible to
the laptop user
      CheckObjectVisibility (LaptopNodeId, ObjectName,
PrimaryRowId)
      IF visible THEN
       DownloadObjectInstance (LaptopNodeId,
                               RelatedDockingObject,
newPrimaryRowId);
      END IF;
    END LOOP;
  END LOOP;

  return TRUE;
}


-- Remove an Object Instance to a Laptop
-- This function also removes all Related Docking Object
```

We claim:

1. A method of collecting, storing, and retrieving data in a database management system having a master database server (4), at least one workgroup server (315), and a plurality of workgroup user clients (310), said workgroup server (315) interposed between said master database server (4) and said workgroup user clients (310) comprising:
   (a) creating a transaction and a transaction file corresponding thereto in one of said workgroup user clients (310); and
   (b) updating said transaction file into a database resident on said workgroup server (315).

2. The method of claim 1 comprising:
   (a) creating a transaction in a local database resident on one of said workgroup user clients (310), entering the transaction into a transaction log resident on said workgroup user client (310), and creating a transaction file corresponding thereto in an outbox of said workgroup user client (310);
   (b) copying said transaction file to an inbox identified to the workgroup user client (310) and updating said transaction file into a workgroup database (305) resident on said workgroup server (315), said workgroup database (305) including a transaction log; and
   (c) reading said workgroup database (305) transaction log, skipping those transactions which originate at the master database server (4), creating data files corresponding to the entries therein, copying data files corresponding to transactions originating at the workgroup user client (310) to

and updating the transactions into a master database (3) on the master database server (4).

6. The method of claim 4 comprising:

(a) copying said transaction file to an inbox identified to the workgroup user client (310) and updating said transaction file into a workgroup database (305) resident on said workgroup server (315), said workgroup database (305) including a transaction log; and

(b) reading said workgroup database (305) transaction log, skipping those transactions which originate at the master database server (4), creating data files corresponding to the entries therein, copying data files corresponding to transactions originating at the workgroup user client (310) to an inbox on the master database server (4) corresponding to the workgroup server (315), and updating the transactions into a master database (3) on the master database server (4).

7. An article of manufacture comprising a computer usable medium having computer readable program code means embodied therein for collecting, storing, and retrieving data in a database management system having a master database server (4), at least one workgroup server (315), and a plurality of workgroup user clients (310), said workgroup server (315) interposed between said master database server (4) and said workgroup user clients (310), the computer readable program means in said article of manufacture comprising:

(a) computer readable program code means for causing a computer to effect creating a transaction in a local database resident on one of said

steps comprising:

(a) creating a transaction in a local database resident on one of said workgroup user clients (310), entering the transaction into a transaction log resident on said workgroup user client (310), and creating a transaction file corresponding thereto in an outbox of said workgroup user client (310);

(b) copying said transaction file to an inbox identified to the workgroup user client (310) and updating said transaction file into a workgroup database (305) resident on said workgroup server (315), said workgroup database (305) including a transaction log; and

(c) reading said workgroup database (305) transaction log , skipping those transactions which originate at the master database server (4), creating data files corresponding to the entries therein, copying data files corresponding to transactions originating at the workgroup user client (310) to an inbox on the master database server (4) corresponding to the workgroup server (315), and updating the transactions into a master database (3) on the master database server (4).

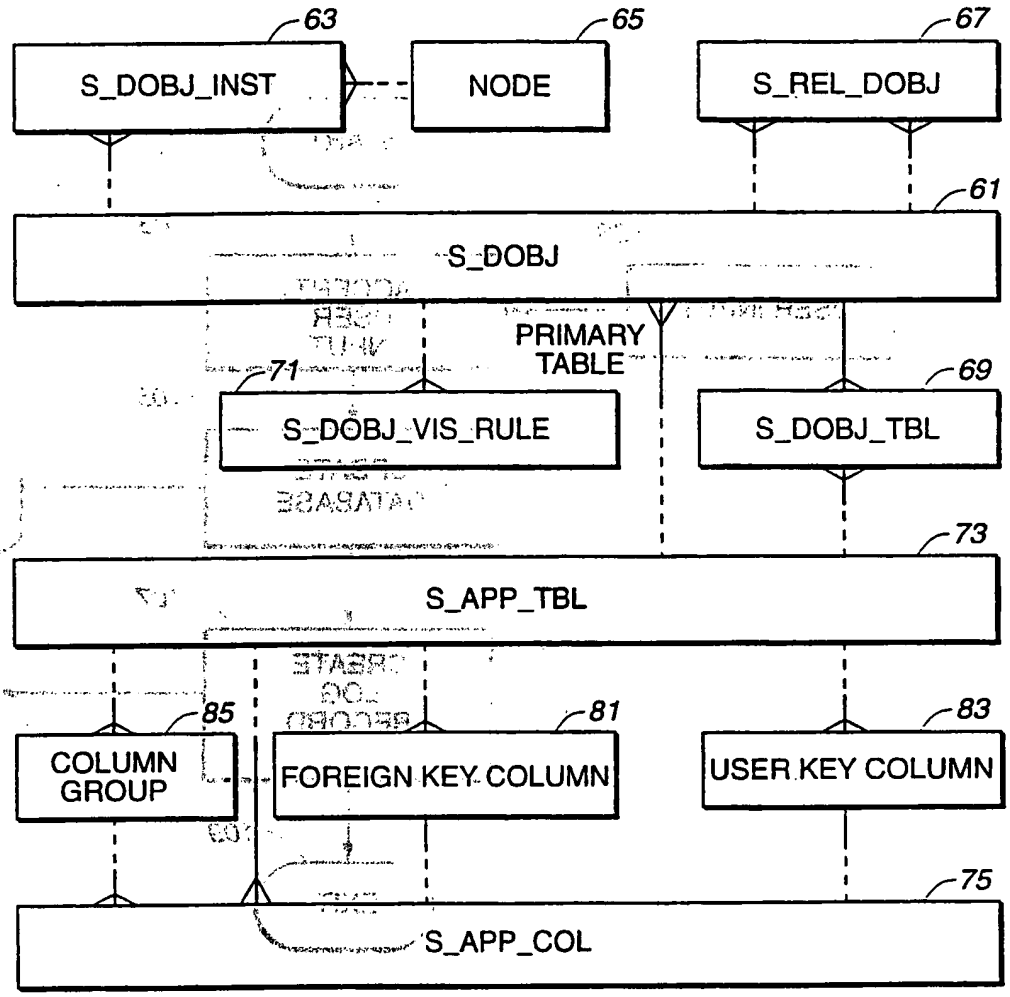

FIG._2

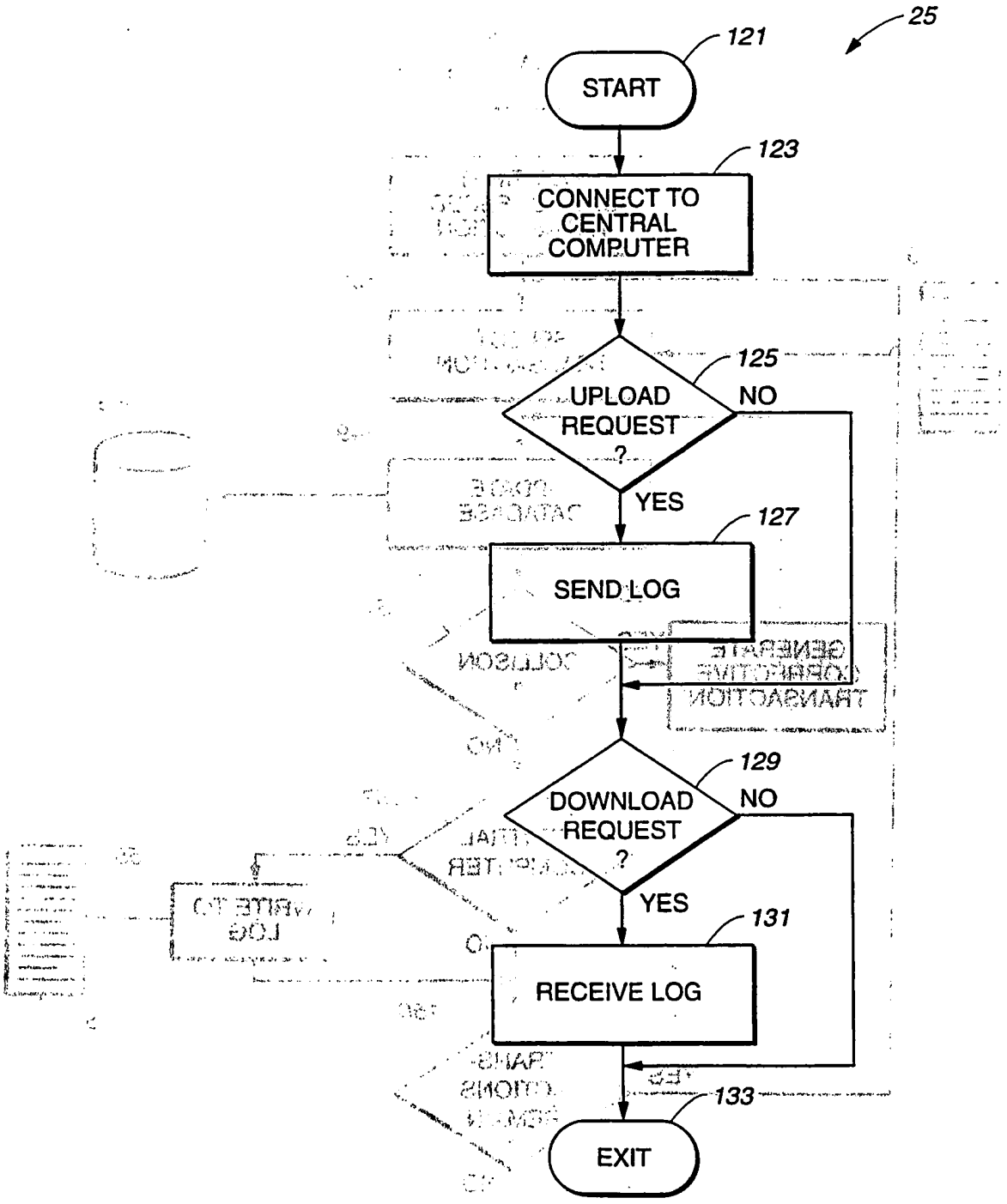

FIG._4

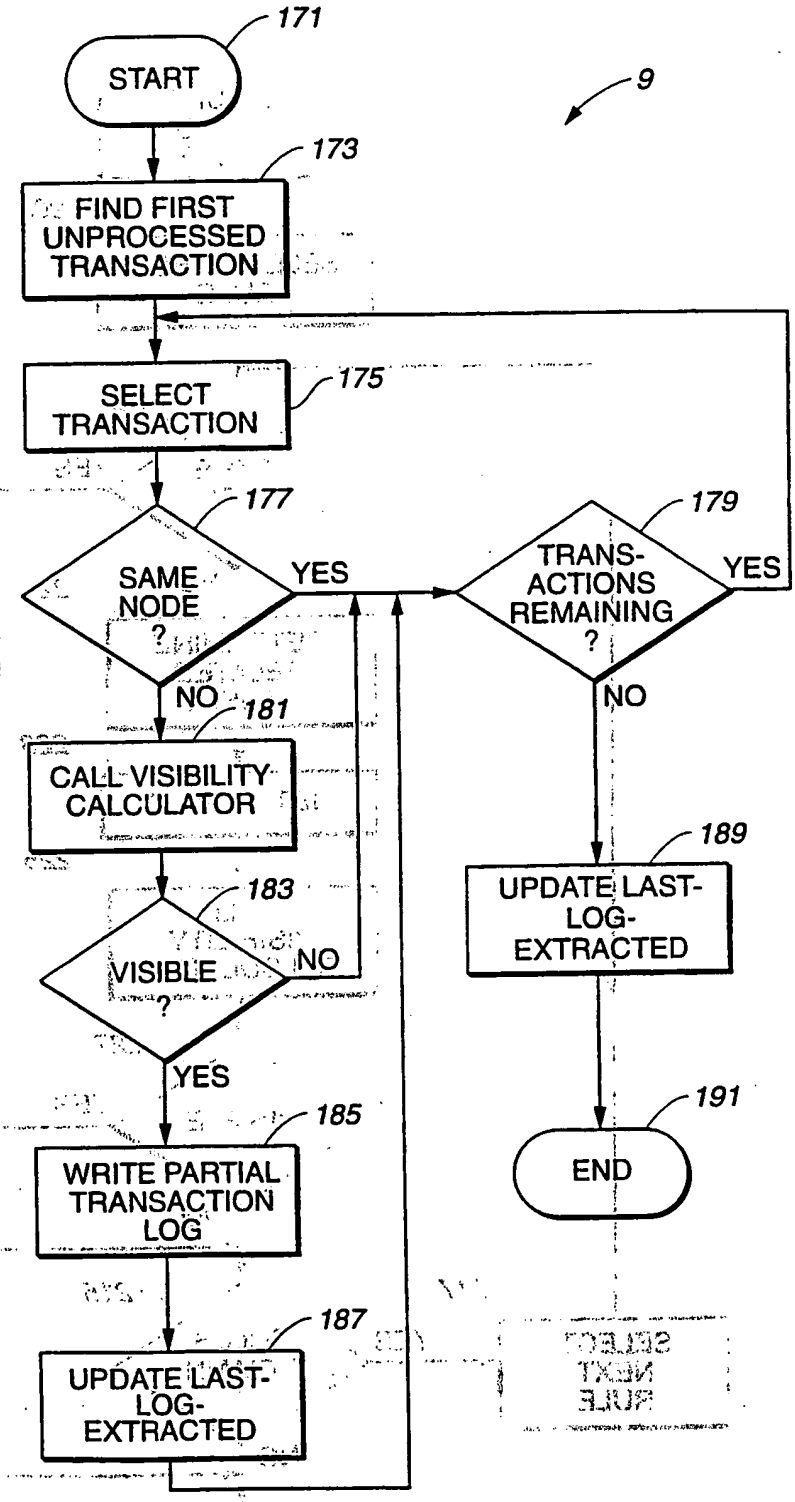

FIG._6

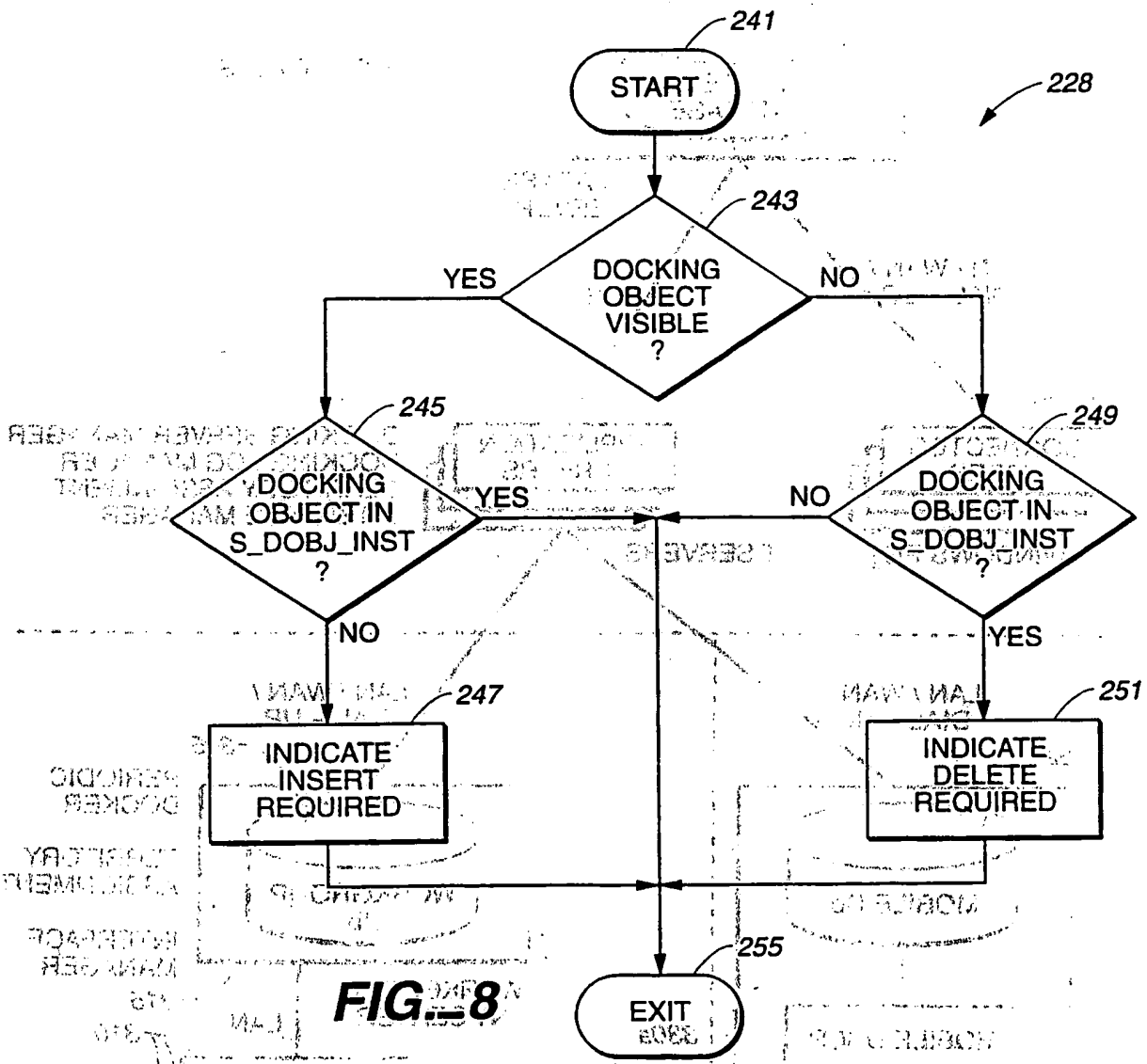

FIG. 8

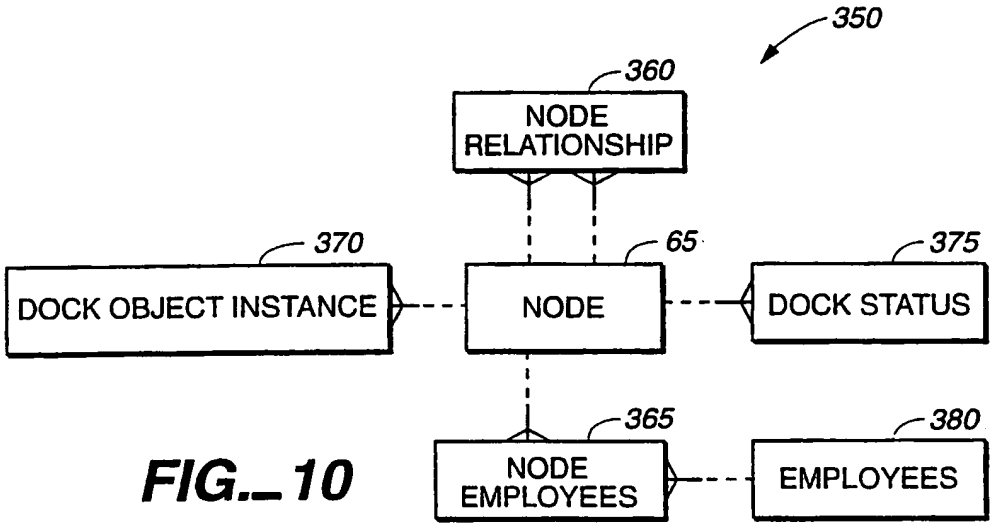
350
360
NODE RELATIONSHIP
370
DOCK OBJECT INSTANCE
65
NODE
375
DOCK STATUS
365
NODE EMPLOYEES
380
EMPLOYEES

FIG._10

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 17/30 — A3

(11) International Publication Number: WO 98/38564

(43) International Publication Date: 3 September 1998 (03.09.98)

(21) International Application Number: PCT/US98/03752

(22) International Filing Date: 24 February 1998 (24.02.98)

(30) Priority Data:
60/039,230 28 February 1997 (28.02.97) US

(71) Applicant *(for all designated States except US)*: SIEBEL SYSTEMS, INC. [US/US]; 1855 South Grant Street, San Mateo, CA 94402 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: BRODERSEN, Robert, S. [US/US]; 17 Spinaker Drive, Redwood City, CA 94065 (US). CHATTERJEE, Prashant [US/US]; 21281 Glenmont Drive, Saratoga, CA 95070 (US). LIM, Peter, S. [US/US]; 917 Governors Bay Drive, Redwood City, CA 94065 (US).

(74) Agents: GOLDMAN, Richard, M.; Cooley Godward LLP, Five Palo Alto Square, 3000 El Camino Real, Palo Alto, CA 94306-2155 (US) et al.

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, GW, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(88) Date of publication of the international search report:
29 October 1998 (29.10.98)

(54) Title: PARTIALLY REPLICATED DISTRIBUTED DATABASE WITH MULTIPLE LEVELS OF REMOTE CLIENTS

(57) Abstract

A method of and system for collecting, storing, and retrieving data in a database management system. The database management system includes a master database server (4), at least one workgroup server (315), and a plurality of workgroup user clients (310). The workgroup server (315) is interposed between the master database server (4) and said workgroup user clients (310). The method creating a transaction in a local database resident on one of the workgroup user clients (310), entering the transaction into a transaction log resident on the workgroup user client (310), and creating a transaction file corresponding to the transaction in an outbox of said workgroup user client (310). Next, the transaction file is copied to an inbox identified to the workgroup user client (310) and updating the transaction file into a workgroup database (305) resident on the workgroup server (315). The workgroup database (305) includes a transaction log.

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US98/03752

A. CLASSIFICATION OF SUBJECT MATTER
IPC(6) :G06F: 17/30
US CI. :707/1, 10, 100, 101, 103
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 707/1, 10, 100, 101, 103

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

none

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS
workgroups, groupware, middleware, shareware, database distribution, file accessing, transaction management, logging

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X,E	US 5,764,899 A (EGGLESTON et al.) 09 June 1998 (09.06.98), abstract, lines 1-16, fig. 1, items 110 & 115fig. 2, items 201-215 & 226-264.	1-6
Y,E	US 5,765,038 A (FLANNERY et al.) 09 June 1998 (09.06.98), col. 9, claim 1, lines 25-64, col. 10, claims 2-8, & lines 1-62.	7-8
Y,P	US 5,664,183 A (CIRULLI et al.) 02 September 1997 (02.09.97), abstract, lines 1-10, col. 9, claim 1, lines 23-42, col. 9, claims 9 & 10, lines 6-65, & col. 11, claim 13, lines 7-38.	1-8
A, E	US 5,745,897 A (PERKINS et al.) 28 April 1998 (28.04.98), abstract, lines 1-19.	1-8

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search
23 JUNE 1998

Date of mailing of the international search report
27 AUG 1998

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703) 305-3230

Authorized officer
CHERYL LEWIS Diane Smith for
Telephone No. (703) 305-8750

Form PCT/ISA/210 (second sheet)(July 1992)*

BNSDOCID: <WO___9838584A3_I_>